LA-UR-17-30438
Approved for public release; distribution is unlimited.

Title:	Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC
Author(s):	Randles, Timothy C. Priedhorsky, Reid
Intended for:	Supercomputing 2017, 2017-11-13 (Denver, Colorado, United States)
Issued:	2017-11-14

Charliecloud

Unprivileged Containers for User-Defined Software Stacks in HPC

Tim Randles
Reid Priedhorsky

November 15, 2017

NNSA
National Nuclear Security Administration
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

The next 25 minutes of your life

1. Why user-defined software stacks will end your suffering
2. But only if you use containers
3. Use Charliecloud and all your wildest dreams will come true

Some people need different software

Default software stacks are good at specific things.

– in the case of HPC, it's MPI-based simulation codes

What if your thing is different?

– non-MPI simulations
– data analytics and machine learning
– epic build process

Admins will install software for you.

– BUT only if there's enough demand
– unusual needs go unmet
– are you crackpot or innovative?

Solution: User-defined software stacks

BYOS (bring your own software)

- Let users install software of their choice
- ... up to and including a complete Linux distribution
- ... and run this image on compute resources they don't own.

Why User-Defined Software Stacks (UDSS)?

Advantages

- software dependencies: numerous, unusual, older, newer, internet ...
- portability of environments: e.g., across dev/test/small/large ...
- consistent environments: validated, standardized, archival ...
- usability

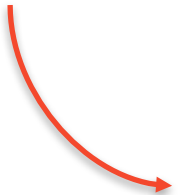
Why User-Defined Software Stacks (UDSS)?

Advantages

- software dependencies: numerous, unusual, older, newer, internet ...
- portability of environments: e.g., across dev/test/small/large ...
- consistent environments: validated, standardized, archival ...
- usability

Disadvantages (possibly)

- missing functionality: HSN, accelerators, file systems
- performance: many opportunities for overhead



Design goals
1. Standard, reproducible workflow
2. Work well on existing resources
3. Be very simple

Design goals

1. **Standard, reproducible workflow**
   - in contrast with “tinker ’til it’s ready, then freeze”
   - standard ⟹ reduce training/devel costs, increase skill portability
   - reproducible ⇒ creation of images is easier & more robust
2. **Work well on existing resources**
   - HPC centers are very good at what they do
   - let’s not re-implement and re-optimize
     - resource management: solved (Slurm, Moab, Torque, PBS, etc.)
     - file systems: solved (Lustre, Panasas, GPFS)
     - high-speed interconnect: solved (InfiniBand, OPA)
3. **Be very simple**
   - save costs: development, debugging, security, usability
   - UNIX philosophy: “make each program do one thing well”

UDSS Options

options	definition	UDSS shares with host…			pros	cons
		kernel	core libraries	app libraries		

UDSS Options

options	definition	UDSS shares with host…			pros	cons
		kernel	core libraries	app libraries		
compile it yourself	download all your dependencies and compile them	yes	yes	mixed	always available; in principle can do anything	not 1995 anymore; in practice too hard

UDSS Options

options	definition	UDSS shares with host…			pros	cons
		kernel	core libraries	app libraries		
compile it yourself	download all your dependencies and compile them	yes	yes	mixed	always available; in principle can do anything	not 1995 anymore; in practice too hard
virtual machines	program (software) that emulates a computer (hardware)	no	no	no	maximum flexibility and isolation	too heavyweight; HPC is not cloud

UDSS Options

options	definition	UDSS shares with host…			pros	cons
		kernel	core libraries	app libraries		
compile it yourself	download all your dependencies and compile them	yes	yes	mixed	always available; in principle can do anything	not 1995 anymore; in practice too hard
virtual machines	program (software) that emulates a computer (hardware)	no	no	no	maximum flexibility and isolation	too heavyweight; HPC is not cloud
containers	isolate UDSS using kernel mechanisms	yes	optional	optional	easy to manage; good performance; *sufficient* flexibility and isolation	new

UDSS Options

options	definition	UDSS shares with host… kernel	core libraries	app libraries	pros	cons
compile it yourself	download all your dependencies and compile them	yes	yes	mixed	always available; in principle can do anything	not 1995 anymore; in practice too hard
virtual machines	program (software) that emulates a computer (hardware)	no	no	no	maximum flexibility and isolation	too heavyweight; HPC is not cloud
containers	isolate UDSS using kernel mechanisms	yes	optional	optional	easy to manage; good performance; *sufficient* flexibility and isolation	new

Container implementations

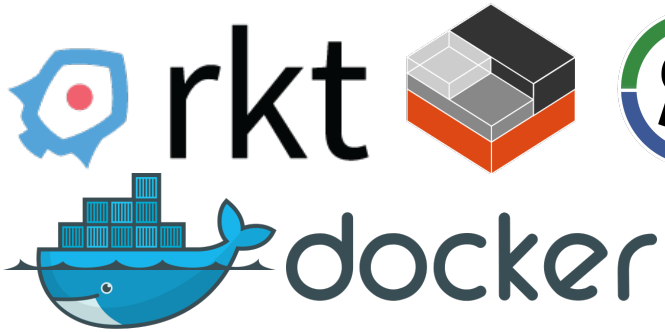
Full-featured

- image building
- image management
  - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
  - e.g., default command/script, inetd-alike
- stateful containers
- supervisor daemon(s)

Container implementations

Full-featured

- image building
- image management
  - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
  - e.g., default command/script, inetd-alike
- stateful containers
- supervisor daemon(s)

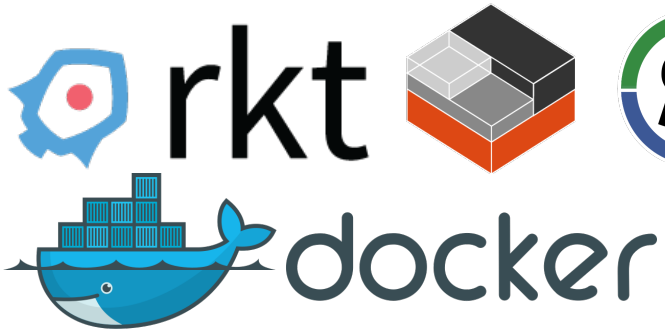


systemd-nspawn [???]
NsJail [???]

Container implementations

Full-featured

- image building
- image management
  - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
  - e.g., default command/script, inetd-alike
- stateful containers
- supervisor daemon(s)



systemd-nspawn [???]
NsJail [???]

Features are useful, but drawbacks…

1. code size
2. support burden
3. privileged & trusted operations

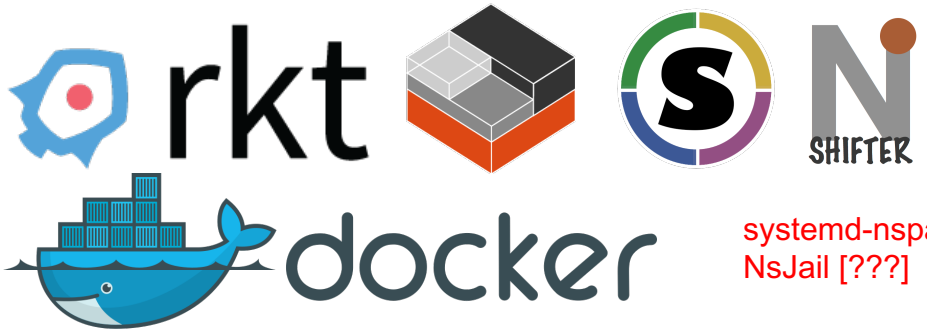
Container implementations

Full-featured

- image building
- image management
  - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
  - e.g., default command/script, inetd-alike
- stateful containers
- supervisor daemon(s)

Lightweight

- few features
- given an image, run it



systemd-nspawn [???]
NsJail [???]

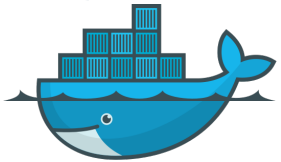
Features are useful, but drawbacks…

1. code size
2. support burden
3. privileged & trusted operations

Container implementations

Full-featured

- image building
- image management
  - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
  - e.g., default command/script, inetd-alike
- stateful containers
- supervisor daemon(s)



systemd-nspawn [???]
NsJail [???]

Features are useful, but drawbacks…

1. code size
2. support burden
3. privileged & trusted operations

Lightweight

- few features
- given an image, run it

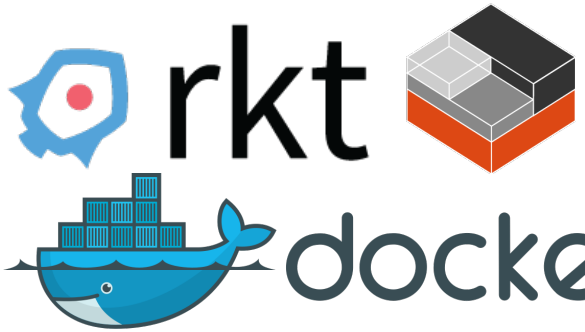
`unshare(1)`
systemd-nspawn [???]
NsJail [???]
Charliecloud

Lower-cost deployment

Container implementations

Full-featured

- image building
- image management
  - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
  - e.g., default command/script, inetd-alike
- stateful containers
- supervisor daemon(s)



systemd-nspawn [???]
NsJail [???]

Features are useful, but drawbacks…

1. code size
2. support burden
3. privileged & trusted operations

Lightweight

- few features
- given an image, run it

`unshare(1)`
systemd-nspawn [???]
NsJail [???]
Charliecloud

Lower-cost deployment

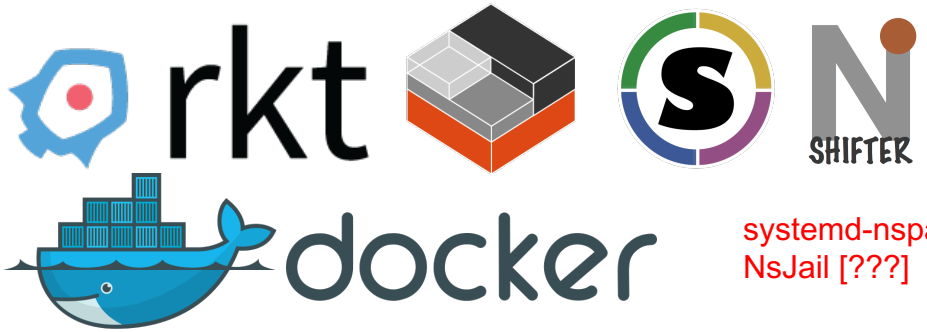
Conclusion: Lightweight implementations are a better choice for HPC centers

- most important cloud-like flexibility
- don't compromise existing tools & strengths of HPC centers

Container implementations

Full-featured

- image building
- image management
  - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
  - e.g., default command/script, inetd-alike
- stateful containers
- supervisor daemon(s)



systemd-nspawn [???]
NsJail [???]

Features are useful, but drawbacks…

1. code size
2. support burden
3. privileged & trusted operations

Lightweight

- few features
- given an image, run it

`unshare(1)`
systemd-nspawn [???]
NsJail [???]
Charliecloud

Lower-cost deployment

Conclusion: Lightweight implementations are a better choice for HPC centers

- most important cloud-like flexibility
- don't compromise existing tools & strengths of HPC centers

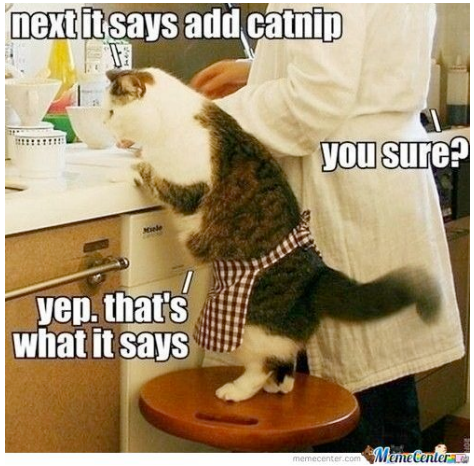
But … some of those other features are important

Container ingredients to mix-n-match

1. **Linux namespaces**
   - **mount**: filesystem tree and mounts
   - **PID**: process IDs
   - **UTS**: host name & domain name
   - **network**: all other network stuff
   - **IPC**: System V and POSIX
   - **user**: UID/GID/capabilities

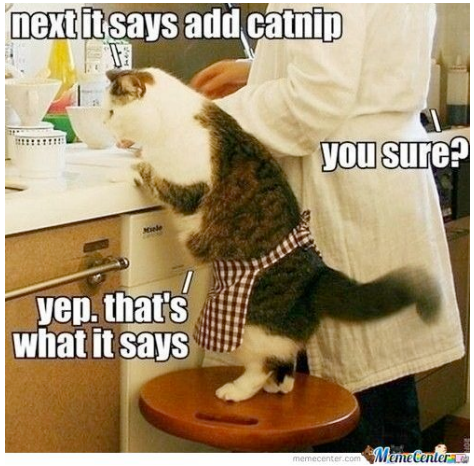
Container ingredients to mix-n-match

1. **Linux namespaces**
   - **mount**: filesystem tree and mounts
   - **PID**: process IDs
   - **UTS**: host name & domain name
   - **network**: all other network stuff
   - **IPC**: System V and POSIX
   - **user**: UID/GID/capabilities
2. **cgroups**
   - limit resource consumption per process
3. **prctl (PR_SET_NO_NEW_PRIVS)**
   - prevent `execve(2)` from increasing privileges
4. **seccomp(2)**
   - filter system calls
5. **SELinux, AppArmor, etc.**
   - various features that change what a process may do



Container ingredients to mix-n-match

1. **Linux namespaces** ← System calls: `unshare(2)`, `clone(2)`, `setns(2)`
   - **mount**: filesystem tree and mounts
   - **PID**: process IDs
   - **UTS**: host name & domain name
   - **network**: all other network stuff
   - **IPC**: System V and POSIX
   - **user**: UID/GID/capabilities
2. **cgroups**
   - limit resource consumption per process
3. **prctl (PR_SET_NO_NEW_PRIVS)**
   - prevent `execve(2)` from increasing privileges
4. **seccomp(2)**
   - filter system calls
5. **SELinux, AppArmor, etc.**
   - various features that change what a process may do

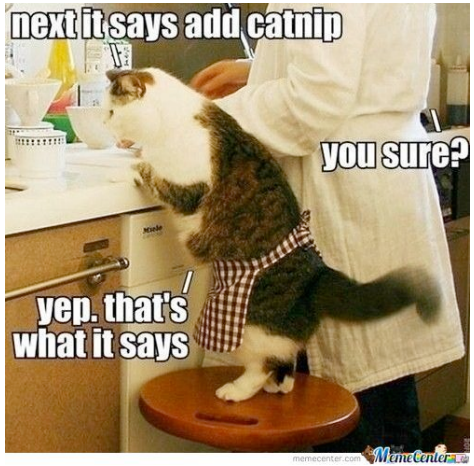


Container ingredients to mix-n-match

1. **Linux namespaces**
   - **mount**: filesystem tree and mounts
   - **PID**: process IDs
   - **UTS**: host name & domain name
   - **network**: all other network stuff
   - **IPC**: System V and POSIX
   - **user**: UID/GID/capabilities
2. **cgroups**
   - limit resource consumption per process
3. **prctl (PR_SET_NO_NEW_PRIVS)**
   - prevent `execve(2)` from increasing privileges
4. **seccomp(2)**
   - filter system calls
5. **SELinux, AppArmor, etc.**
   - various features that change what a process may do

System calls: `unshare(2)`, `clone(2)`, `setns(2)`

privileged
need root to create, unless you add…



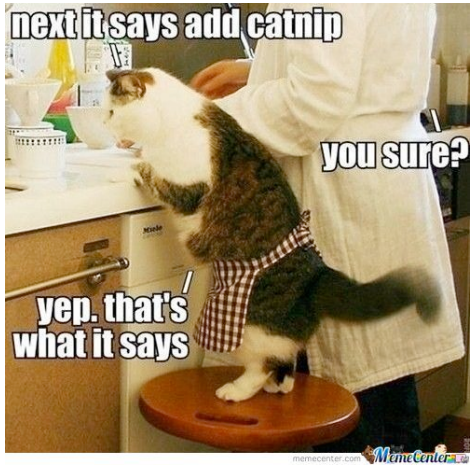
Container ingredients to mix-n-match

1. **Linux namespaces**
   - **mount**: filesystem tree and mounts
   - **PID**: process IDs
   - **UTS**: host name & domain name
   - **network**: all other network stuff
   - **IPC**: System V and POSIX
   - **user**: UID/GID/capabilities
2. **cgroups**
   - limit resource consumption per process
3. **prctl (PR_SET_NO_NEW_PRIVS)**
   - prevent `execve(2)` from increasing privileges
4. **seccomp(2)**
   - filter system calls
5. **SELinux, AppArmor, etc.**
   - various features that change what a process may do

System calls: `unshare(2)`, `clone(2)`, `setns(2)`

privileged
need root to create, unless you add…

unprivileged



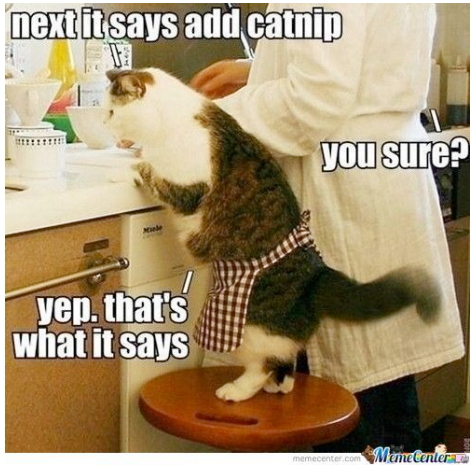
Container ingredients to mix-n-match

1. **Linux namespaces**
   - **mount**: filesystem tree and mounts
   - **PID**: process IDs
   - **UTS**: host name & domain name
   - **network**: all other network stuff
   - **IPC**: System V and POSIX
   - **user**: UID/GID/capabilities
2. **cgroups**
   - limit resource consumption per process
3. **prctl (PR_SET_NO_NEW_PRIVS)**
   - prevent `execve(2)` from increasing privileges
4. **seccomp(2)**
   - filter system calls
5. **SELinux, AppArmor, etc.**
   - various features that change what a process may do

System calls: `unshare(2)`, `clone(2)`, `setns(2)`

privileged
need root to create, unless you add…

unprivileged



Charliecloud

Photo-illustration by Hunter Easterday, Jordan Ogas, Adam Smith / HPC-DES

Charliecloud's hybrid approach

1. **Image building & sharing goes in a sandbox**
   - safe place for users to be root: user workstation or virtual machine
   - use Docker for image building
     - or anything else that can produce a filesystem tree
     - debootstrap(8), yum --installroot, etc.
   - wrap Docker for image management
     - ch-docker2tar

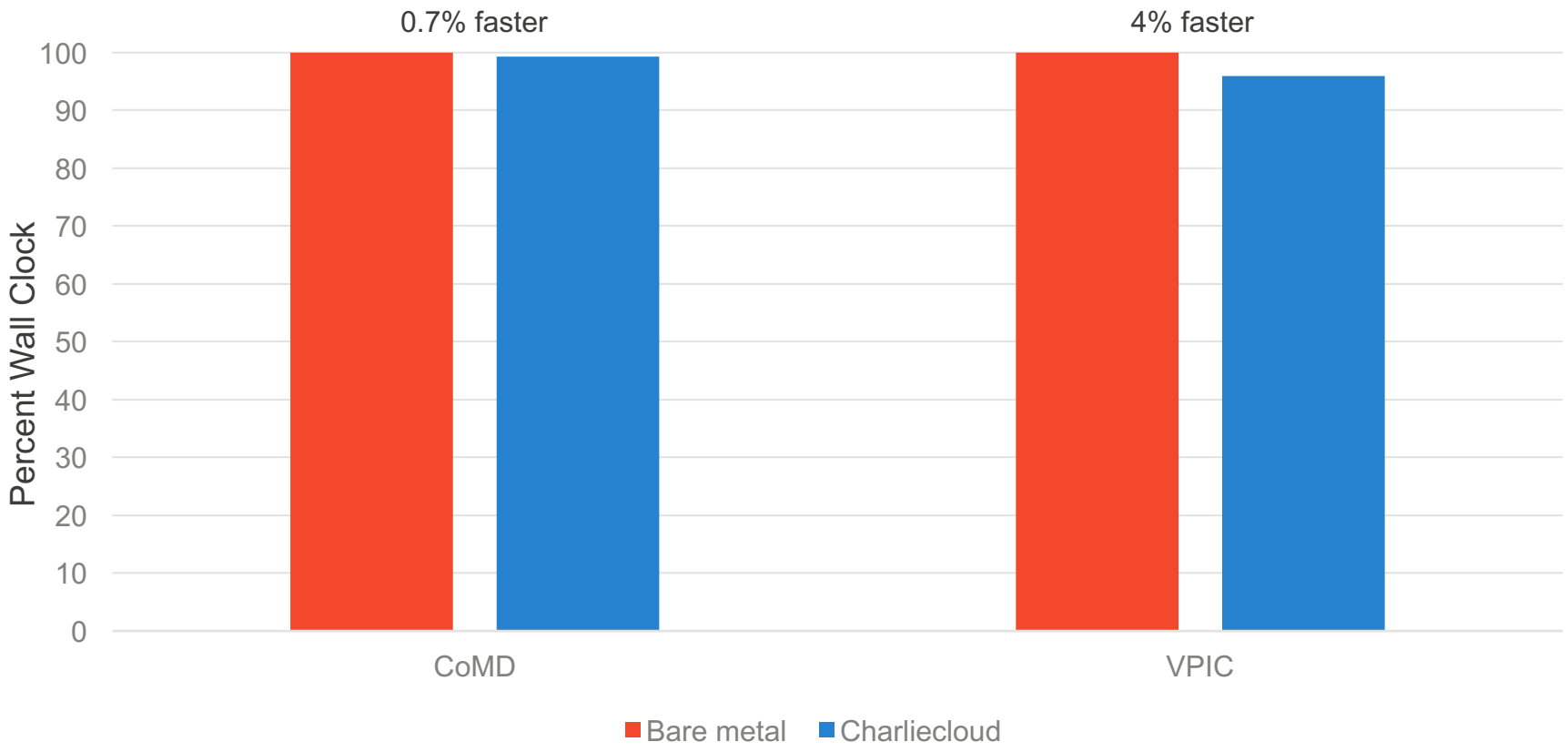
Charliecloud's hybrid approach

1. **Image building & sharing goes in a sandbox**
   - safe place for users to be root: user workstation or virtual machine
   - use Docker for image building
     - or anything else that can produce a filesystem tree
     - debootstrap(8), yum --installroot, etc.
   - wrap Docker for image management
     - ch-docker2tar
2. **Run images with our own unprivileged runtime**
   - mount & user namespaces only
     - requires new-ish kernel
     - most distros have the right kernel (Fedora in 2015, Ubuntu Xenial in 2016)
     - Cray UP04 has it
     - RHEL/CentOS 7 can install via ElRepo (or enable on kernel command line in 7.4)
   - **it's a user program!!!**
   - admins don't need to do anything

Basic workflow

step	where		privileged?
	sandbox	production	
1. Build Docker/etc. image	✓		maybe
2. Dump image to tarball	✓		maybe
3. Copy tarball to where you want to run	✓	✓	no
4. Unpack tarball		✓	no
5. Configure your stuff (sometimes)		✓	no
6. Run your commands in container		✓	no

Performance e.g.: CoMD and VPIC (32 nodes)



Charliecloud vs. the design goals

✓ **1. Standard, reproducible workflow**

- in contrast with “tinker ’til it’s ready, then freeze”
- standard ⟹ reduce training/devel costs, increase skill portability
- reproducible ⇒ creation of images is easier & more robust

✓ **2. Work well on existing resources**

- HPC centers are very good at what they do
- let’s not re-implement and re-optimize

  resource management: solved (Slurm, Moab, Torque, PBS, etc.)

  file systems: solved (Lustre, Panasas, GPFS)

  high-speed interconnect: solved (InfiniBand, OPA)

✓ **3. Be very simple**

- save costs: development, debugging, security, usability
- UNIX philosophy: “make each program do one thing well”

Charliecloud status

1. **Available now on some LANL clusters**
   - passes tests on Crays
   - Woodchuck (IC) now, Fog (ASC) very soon
2. **Installable now on any Linux box**
   - newer kernel needed (roughly 4.4+)
   - including cloud instances
3. **Instructions for pre-installed VirtualBox image**
   - no root needed
   - Mac, Windows, Linux, Solaris
4. **Packages available on openSUSE Build Service (community)**
   - CentOS 7, Debian 9.0, Xubuntu 16.04 & 17.10
5. **PR for HTCondor integration (community)**

Charliecloud resources

;login: article (USENIX magazine)

- "Linux containers for fun and profit in HPC"
- https://www.usenix.org/publications/login/fall2017/priedhorsky

Supercomputing 2017

- "Charliecloud: Unprivileged containers for UDSS in HPC"

Documentation

- https://hpc.github.io/charliecloud
- includes detailed tutorials

Source code

- https://github.com/hpc/charliecloud

Reid Priedhorsky, Tim Randles / {reidpr,trandles}@lanl.gov

Charliecloud: Lightweight unprivileged containers for UDSS in HPC